Algoritmo Grasp para la distribución eficiente de objetos en una interfaz gráfica de usuarios

Juan Zamudio[1], Luis Rivera[2], David Mauricio[1,3]

[1] Universidad Nacional Mayor de San Marcos, FISI
Av. German Amezaga s/n Lima 1, Lima, Perú

[2] Universida de Estadual do Norte Fluminense, LCMAT-CCT,
Av. Alberto Lamego 2000, Campos dos Goytacazes, Rio de Janeiro, Brasil, 28015-620

[3] Universidad Inca Garcilaso de la Vega, FISCT
Av. Bolívar 1848 – Lima 21, Lima, Perú

Jzamudio@unmsm.pe, rivera@uenf.br, dms_research@yahoo.com

Abstract The technical evolution allowed in the last years that the human-computer interaction can be, in good part, using graphical user interface composed by communication objects. The success of that interaction depends, in good part, of the best distribution of the communication objects on the interface. The distribution of the objects on the interface is, then, one of the principal parameter to define efficiency of the interface. In this work, we considered that the efficiency of the graphic user interface depends directly on the time used in the accomplishment of a task by an user operating an software applicative, therefore will convenience a best distribution of the objects for minimized the time used in execute a task. We propose a method based on Heuristic Grasp to design an efficient distribution of objects for an interface.
Keywords: Grasp, GUI, Graphic User Interface, GUI Validation, GUI Efficiency.

Abstrac

Resumen La evolución técnica permitió en los últimos años que la comunicación usuario-computadora sea, en buena parte, a través de una interfaz gráfica de usuarios compuesta por objetos de comunicación. El éxito de esa interacción depende, en buena parte, de la mejor forma de distribución de los objetos de comunicación en la interfaz. La distribución de los objetos en la interfaz es, entonces, uno de los parámetros principales que da atributo de eficiencia a la interfaz. En este trabajo, consideramos que la eficiencia de la interfaz gráfica de usuario depende directamente del tiempo usado en la realización de una tarea por un usuario operando un aplicativo, por tanto será conveniente una mejor distribución de los objetos para minimizar el tiempo de la realización de una tarea. Proponemos un método basado en la metaheurística Grasp para diseñar una distribución eficiente de los objetos de una interfaz.
Palabras Claves: Grasp, GUI, Interfaz Gráfica de Usuario, Validación de una GUI, GUI eficiente.

Resumen

1. Introducción

Los programas que requieren de la intervención humana para realizar las tareas para las cuales fueron diseñadas, sea para su interacción de controles, entrada de datos o simplemente ver los resultados y mensajes, deben ofrecer al usuario humano un ambiente de comunicación llamada interfaz, compuesto por una distribución de objetos (textos, botones, ventanas, imágenes, mensajes, etc.) visualmente entendibles, generalmente gráficos. Esos elementos de comunicación, conocido en los tiempos actuales como Interfaz Gráfica de Usuario (Graphic User Interface - GUI), deben ser amigables del punto de vista del usuario, simples de entendimiento y eficientes en la interacción.

La simplicidad permite que cualquier usuario pueda interactuar con el programa sin demandar tiempo y esfuerzo para familiarizarse con la utilización del programa. La amigabilidad permite que el usuario se sienta ameno y atraído por la interfaz para su manipulación, para que de este modo no sienta rechazos y otros efectos negativos de la interfaz. Eficiencia hace que el usuario no requiera mucho esfuerzo, o pierda tiempo en realizar un determinado trabajo interactuando con los elementos de control de la interfaz.

Muchas veces, una mala distribución de los objetos de la interfaz puede demandar más tiempo de ejecución de un determinado trabajo con el programa; por ejemplo, realizar un trabajo que demande cuatro tareas: capturar datos, operar, mostrar las variaciones y salvar los resultados. Cada tarea está asociada a un objeto de control, tipo botón. El usuario puede realizar ese trabajo en un determinado tiempo si los objetos de control están ordenados en la secuencia de la operación de la tareas, mientras que el tiempo demorado en realizar ese mismo trabajo será diferente, posiblemente mayor, si los objetos están posiciones arbitrariamente sobre la interfaz. Esa forma de realizar un trabajo puede generar, a parte de la demora, efectos de cansancio tanto físico y mental en el usuario, como también mayor actividad computacional debido a los movimientos redundantes del mouse o cualquier elemento de activación de los objetos de la interfaz.

Una forma de ver que una GUI es apropiada es minimizando el tiempo de ejecución de una sesión de trabajo por un usuario. Consideramos una sesión de trabajo a la realización de una o más tareas. Una tarea es una secuencia de acciones realizadas por el usuario para efectuar un trabajo. En este caso, las acciones son las activaciones de los objetos de la interfaz, así como desplazamiento del foco de acción (mouse, visión, tacto, etc.) de un objeto a otro. Ese desplazamiento lo denominamos como transición. También, podemos considerar en las acciones el grado de dificultad de localización del siguiente objeto a ser activado, que puede demandar un esfuerzo al usuario si esos objetos no están explícitamente definidos (con significados propios), y no están eficientemente ubicados respecto al objeto anterior.

En este trabajo, proponemos un método eficiente de redistribución de los objetos de la interfaz de un programa computacional. El método es basado en la meta-heurística Grasp [1]. Los resultados computacionales demuestran una eficiencia superior respecto a los métodos existentes.
El resto del trabajo está organizado de la siguiente forma: en la Sección 2 damos una revisión de los trabajos previos; en la Sección 3 presentamos una estructura de redistribución de los objetos mostrando los indicadores apropiados de una interfaz eficiente; en la Sección 4 formulamos el método usando la meta-heurística Grasp en la eficiencia de la GUI propuesta; la Sección 5 muestra los resultados computacionales y discusiones; y, finalmente, en la Sección 6 concluimos.

2. Trabajos previos

En orden de mejorar cada vez la interfaz gráfica de usuario, surgen varios enfoques relacionados con diseño de interfaz gráfica, casi todos con principios de usabilidad. Así, por ejemplo Cusner y Lakin [3] desarrollan un sistema que realiza exhibición gráfica de descripción de las tareas, donde los operadores visuales son sustituidos automáticamente por los operadores lógicos. Con operadores lógicos se intenta reducir las demandas cognoscitivas puestas en los usuarios. Es un método basado en tareas independientes, por lo tanto no puede evaluar un plan de tal manera que los usuarios de dichas tareas la realicen normalmente.

Goms, formulado por Kieras [4], es un método que demanda descripción detallada de las tareas. Las descripciones pueden ser difíciles de obtenerse, y los cambios en las tareas pueden requerir un análisis bastante extenso. Este tipo de método, puede proporcionar información útil, pero no pueden evaluar la adecuación de una interfaz para las tareas que los usuarios realmente realizan.

Tullis [5] analiza métodos de construcción de GUI prediciendo el despliegue de los usuarios, explorando la relación entre varias métricas, que incluye densidad global de la pantalla, la densidad de la pantalla agrupada de objetos, la complejidad del diseño y el tiempo que los usuarios demoran en extraer la información. No se toma en cuenta la descripción de las tareas que los usuarios realizan, con eso él no pudo

evaluar la adecuación de la interfaz para las tareas que los usuarios realmente realizan.
Perlman [6] desarrolla un modelo axiomático de presentación de la información, para permitir ver a los diseñadores la relación que existe entre los datos de entrada y la información a ser presentada. Incorpora un prototipo que proporciona la regeneración de la interfaz frente a los problemas potenciales que se presentan. El problema de este método es que no hay un mecanismo que compare los diseños alternativos, la única contribución es que este método empieza con las "intenciones del diseñador", no usa descripción de las tareas del usuario a las intenciones del diseñador.

Lohse [7] propone un modelo para predecir el tiempo necesario para extraer la información de un objeto. El trabajo se enfoca sobre las actividades que los usuarios realizan y las tareas cognoscitivas que requieren cada uno para predecir el tiempo necesario para realizar una tarea o un conjunto de tareas. Una vez que estos pasos se completan, se evalúan la facilidad de aprender la interfaz propuesta. El problema es que demanda tiempo considerable esa operación que hace casi impracticable su uso.

Sears [2] introduce el concepto de "diseño apropiado" para calificar de bueno al diseño propuesto cuando este es próximo al diseño óptimo. El diseño propuesto es preparado por el diseñador y el diseño óptimo es hallado redistribuyendo los objetos de la interfaz usando el algoritmo de búsqueda en árbol, basado en el método branch and bound. Para eso, el diálogo es convertido en una grilla con una determinada cantidad de celdas uniformes, que representan una posible posición para cada uno de los objetos visuales. Este modelo incorpora las tareas de usuario que se supone dirigirse con el puntero del mouse de un objeto visual para otro. La distancia que recorre el puntero del mouse es multiplicada por la frecuencia, el resultado es el costo de realizar dicha tarea. La interfaz de usuario tiene un número determinado de tareas de usuario, la sumatoria de los costos de estas tareas es el costo total de la interfaz.

El método de redistribución de objetos de Sears explora el principio del método branch and bound, en que cada nodo del árbol es una secuencia de configuraciones (una reubicación de un objeto por vez, en todas las posibles celdas) respecto a la configuración anterior. Así, en cada nodo de la rama existen tantas configuraciones como número celdas posibles de posicionar un objeto. En cada nodo generado se estiman los costos de los supuestos diseños (configuraciones), y se escoge para el branch comenzando por la configuración de menor costo. Esa secuencia es realizada mientras el costo menor alcanzado no es sobrepasado por el costo de la configuración del nivel del branch. El bound, es el costo menor último alcanzado por los branchs anteriores. La heurística usada en ese método, para controlar el branch, es el costo mínimo de cada configuración. Con eso, no es necesario explorar todas las ramificaciones completas del branch. Siendo, el peor de los casos, que genere todas las configuraciones en la mayoría de las ramas. Sears consigue generar diseños apropiados con considerable costo computacional dependiendo del número de objetos y número de las celdas de la grilla.
Entre los productos comerciales conocidos son: AIDE y ESSI PIE 24306. El AIDE [8] se basa en la metodología de Sears, por tanto puede ser lento porque genera recursividad, da prioridad a la preferencia de los diseñadores. Está desarrollado en Java de Sun Microsystem. Este software fue desarrollado por la Escuela de Ciencias de la Computación de la Universidad Depaul en Chicago. ESSI PIE 24304 [9] es basado en la parte funcional de los objetos visuales y en las acciones que estos realizan. Trata de describir los errores en las rutinas, subrutinas y bucles que estos ejecutan. Tiene una característica muy interesante de colocar CheckPoints en el script de los objetos y, de esta manera, realizar un testing personalizado. Pero puede generar una GUI con objetos peor redistribuidos.

El método propuesto en este trabajo, a diferencia de los productos existentes, se basa principalmente en la reducción de tiempos de ejecución de las tareas de usuario, da preferencia a una redistribución eficiente de los objetos sobre la GUI. Para esto, el método de Sears es una buena opción pero evitando generar recursividades en las búsqueda de la mejor configuración de la GUI propuesta.

3. Eficiencia de una GUI

Una GUI eficiente debe permitir la productividad del usuario antes que la productividad de la máquina. En ese sentido, los objetos de la interfaz deben estar eficientemente distribuidos en el diálogo, y los mensajes de ayuda deben ser sencillos y proveer respuestas a los problemas. Los menús y etiquetas de botones deben tener las palabras claves o símbolos con significado del proceso.
El modelo de procesamiento de información que prevalece en la Psicología ha permitido los siguientes principios de diseño de una GUI eficiente:

- La interfaz tendría que compensar las limitaciones humanas, tanto físicas como cognitivas, siempre que sea posible. No obstante, tendría que ser "transparente", no ponerse en el camino de las acciones del usuario o impedir su progreso. Por otra parte, la interfaz no tendría que sobrecargar al usuario con complejidades innecesarias o distraerlo de su labor.

- Los componentes físicos de la interfaz tendrían que ser diseñados ergonómicamente, teniendo presente el confort y la salud del usuario tanto como sus necesidades.
- La interfaz tendría que ser consistente.
- El estilo de interacción no mandado como manipulación directa y menús son preferibles al lenguaje de orden. Como mínimo, el usuario experimetado tendría que tener capacidad de moverse rápidamente a través de las capas de los menús.
- La interfaz tendría de poder tener acciones reversibles.
- La interfaz tendría que estar sujeto a pruebas al principio del diseño del proceso y durante su desarrollo.

El principio más básico de la interfaz sería estar diseñado alrededor de las necesidades del usuario hasta después de que el sistema haya sido completado, atendiéndose de esta manera las restricciones impuestas por el sistema. Luego, la GUI debe ser evaluada con métricas apropiadas de forma de medir su eficiencia respecto a su usabilidad. Generalmente las métricas de evaluación miden la correcta distribución de los objetos en la GUI, considerando los criterios de funcionalidad y significados de los objetos.

3.1. Métrica para la evaluación de una GUI

Se dice que un producto es bueno o malo y hasta sub-clasificado como aceptable, regular, óptimo, etc. dependiendo de algún indicador deducido a partir de las características del producto. En general, esos indicadores son numéricos, o indicadores de tendencias estadísticas, como también indicadores de eficiencia o complejidad en el uso del producto por los usuarios.

En la ingeniería de software no tenemos medidas absolutas como en la física que tiene voltaje, masa, velocidad, etc. En lugar de ello, intentamos obtener un conjunto de medidas indirectas que dan lugar a métricas que proporcionan una indicación de la calidad de algún tipo de representación de software. Como las medidas y métricas del software no son absolutas, estas están abiertas a debate. Fenton [10] dice lo siguiente: La medición es el proceso por el que se asignan números o símbolos a los atributos de las entidades en el mundo real, de tal manera que las definan de acuerdo con unas reglas claramente definidas.

En el caso de la GUI, hay métricas que se basan principalmente en la aceptabilidad del usuario; por ejemplo, una métrica basada en el tiempo que requiere un usuario para aprender a usar la GUI, las encuestas a los usuarios, entre otros.

En este trabajo usamos la métrica usada por Sears [2] para calcular el índice del diseño apropiado (DA) dada por:

$$DA = \frac{CGO}{CGP},$$

donde CGP es el costo del diseño propuesto, y CGO es el costo del diseño óptimo. El CGP es calculado a partir de la GUI propuesta, como el costo total de una sesión definida por una tabla de tareas, como el ejemplo de la Tabla 1 que muestra siete tareas con sus respectivas frecuencias envolviendo cinco objetos (Directory, File, Filename, Botton-OK, Botton-cancel). La frecuencia (f_k) es el número de veces de ejecución de la tarea durante una determinada sesión de trabajo; la frecuencia relativa (F_k) es a frecuencia dividido por la suma de las frecuencias de una sesión; y es el número de tareas que componen una sesión. La Figura 1 representa el diagrama de realización de las tareas en una determina sesión, donde los pesos de los arcos es la frecuencia relativa de la tarea respectiva.

Tarea k	fk	Fk	Trasición
1	5	0.05	Directory-Filename
2	33	0.33	Inicio-File
3	2	0.02	Inicio-Cacel
4	2	0.02	Filename-Ok
5	8	0.08	Inicio-Filename
6	5	0.05	File-Ok
7	45	0.45	Directory-File
T.Sesión	100	1.00	

Tabla 1: Tareas, frecuencias y frecuencias relativas.

El Costo de una tarea se fundamenta en su transición y se puede relacionar con la distancia que el usuario debe recorrer con el mouse de un objeto a otro y el tiempo que demora en realizar este recorrido. Fitts [11] demostró que el tiempo en alcanzar un objetivo esta en función de la distancia que se recorre y del ancho del objeto.

Así, el costo de la GUI propuesta, para tareas, será:

$$CGP = \sum_{k=1}^{t} F_k C_k$$

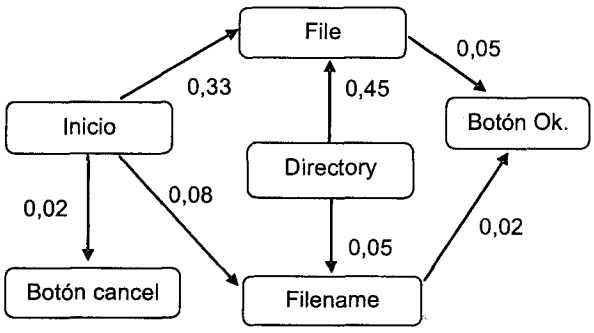

Figura 1. Diagrama de relación de objetos en una sesión.

El costo C_k es la dificultad para realizar una transición por el foco de la acción. Fitts, que propone la primera versión de como calcular C_k, dice: "el tiempo en alcanzar un objetivo está en función de la distancia hacia el objetivo y del ancho del mismo". Así, si el cursor del mouse se desplaza de un punto inicial en una distancia A hacia el objeto O de ancho W en dirección del movimiento del curso, entonces, según Fitts, $C_k = a + b \log_2\left(\frac{2A}{W}\right)$,donde los coeficientes a y b son determinados por métodos estadísticos, en particular por regresión y correlación de rectas y curvas en base a una secuencia de experimentos empíricos. Esa expresión, llamada Ley de Fitts, fue variada por Mackenzie [12] para

$$C_k = a + b \log_2\left(\frac{2A}{W} + 1\right)$$

Las unidades de C_k está en segundos (o milisegundos), ya que a está en segundos (o milisegundos), b está en segundos por bits (o milisegundos por bits) y la variante del índice de dificultad $ID = \log_2\left(\frac{A}{W} + 1\right)$ está dado en bits debido a la base binaria. El hecho que ID tengas unidades en bits es porque cuando el cursor del mouse se mueve de un objeto para otro, el procesador de la computadora procesa bits relacionados con la información en la trayectoria del cursor en la pantalla de la computadora.

Mackenzie realizó un test de viabilidad para determinar los coeficientes de regresión usando cinco modelos: *Smaller of, W', Status Quo, W+H, Area WxH* (mayor detalle ver en [12]). Esos modelos están relacionados con las variaciones de las dimensiones W y H de los objetos destino de transición. Con los valores de test de viabilidad, Mackenzie evaluó los indicadores estadísticos índice de correlación y error estándar. Consideró como válido el valor con bajo error estándar que corresponde al modelo *Smaller of*, asociado a los valores $a = 230$ ms. y $b = 166$ ms/bit. Con esos índices, diremos que el diseño, según (1), es casi óptimo si $DA \approx 1.0$; lo ideal es que $DA = 1.0$ para que sea óptimo.
El valor de *CGO* es el costo de una GUI óptima generada por un algoritmo. En nuestro caso, el diseño óptimo lo generamos usando la meta-heurística Grasp con segmentos del método de Sears.

3.2. Elementos para la generación de una GUI apropiada

Considerando que uno de los principales factores de la eficiencia de una interfaz es la correcta distribución de sus objetos, enfocamos los elementos considerados para construir una interfaz óptima imaginaria a partir de la lista de tareas y la lista de objetos.
Para la generación de una GUI, los objetos que definen la interfaz propuesta deben ser reposicionados de forma a generar una GUI de costo mínimo. En este proceso consideramos el espacio del diálogo de la GUI como el plano Cartesiano, con una grilla homogénea imaginaria en la que son ubicados los objetos. La geometría de los objetos son concebidos como siendo rectangulares (tal como mostrado en la Tabla 2). Para el propósito de este trabajo, consideramos como origen de la coordenada el vértice izquierdo inferior de la grilla, y los ejes numerados por números enteros.

Las dimensiones de las celdas de la grilla están en función de las dimensiones de los objetos. Consideramos la celda de dimensión $MCD_H \times MCD_W$, donde MCD_H y MCD_W son los valores de MCD (máximo común divisor) de las respectivas dimensiones de alto (H) y ancho (W) de los objetos de la interfaz. En la Tabla 2 damos un ejemplo de cinco objetos con sus respectivas dimensiones y
$MCD_H = MCD_W = MCD = 2$

Objetos	Ancho W	Alto H
File	16	8
Directory	16	8
Filename	8	2
Boton OK	4	4
Boton Cancel	2	2
MCD	2	2

Tabla 2. Objetos con sus dimensiones

Para una cuadrícula con n posibles posiciones y m diferentes objetos de representación, el número posible de distribuciones es $n!/(n-m)!$. Cada posible combinación representa un nodo del árbol de búsqueda. Por ejemplo, para $n = 12$ y $m = 5$, el número de nodos posibles generados es 108.384, de los cuales 95.040 son nodos finales que son las posibles configuraciones con la combinación de todos los objetos. Este valor incrementa grandemente cuando el número de objetos incrementa. El método de Sears genera, en mejor de los casos, menor número de nodos, cuando el bound limita la generación de otros branchs. Por ejemplo, para 16 objetos y 5 x 5 celdas, son generados 2.377 nodos de los 1.3×10^{13} posibles nodos y para 28 objetos una grilla de 7 x 6 genera 33.053 nodos de los 1.7×10^{40} posibles nodos.

La Figura 2 representa la GUI propuesta y la Figura 3 es la GUI generada por el método Sears a partir de las tareas definidas en la Tabla 1 envolviendo cinco objetos de la Tabla 2. Los generados costos de cada

GUI son $DGP = 4.39$ y $DGO = 3.92$ respectivamente. Con esos costos, estimados el índice del diseño apropiado $DA = 3.92/4.39 \approx 0.89$.

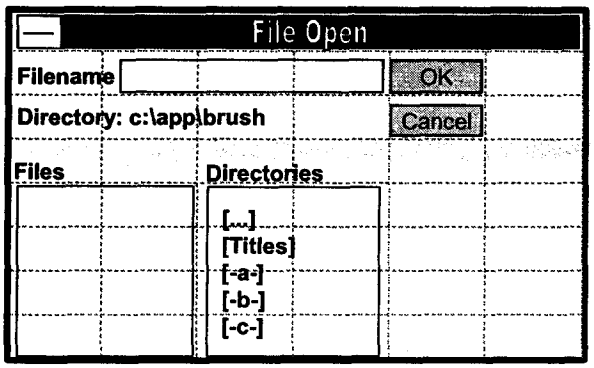

Figura 2. Una GUI propuesta con cinco objetos

En vista que se tienen objetos de dimensiones mayores de una celda, en vez de particionar los objetos en sub-objetos (objetos restringidos) con dimensión igual a la celda, como propuesto por Sears; nosotros agrupamos celdas adyacentes para poder ubicar dichos objetos.

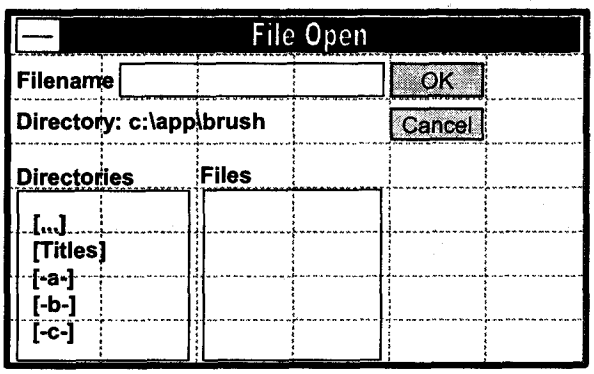

Figura 3. GUI generada por el método Sears

4. Grasp en la generación de una GUI apropiada

La búsqueda de un diseño óptimo está relacionada con la redistribución de los objetos obedeciendo la lista de t tareas de forma que el costo de operación sea mínimo. Para eso, consideremos la función de costos de una configuración enunciada en (2) como la función objetivo:

$$f = \sum_{k=1}^{t} F_k C_k$$

Esa expresión es resuelta, en este trabajo, usando meta-heurística Grasp. Este método nos dará un resultado aproximadamente óptimo por tratarse de un paradigma de construcción aleatoria del mejor resultado.

4.1. Meta-heurísticas Grasp

El método Grasp fue desarrollado al final de la década de los 80 con el objetivo inicial de resolver problemas de cubrimientos de conjuntos [13]. El término Grasp fue introducido por Feo y Resende [1] como una técnica meta-heurística de propósito general.

Un Grasp es un método de multi-arranque, en la cual cada iteración Grasp consiste de dos fases seguidas: construcción golosa-aleatoria de una solución y mejoría de la solución construida a través de alguna técnica de búsqueda local. Este procedimiento se repite varias veces y la mejor solución encontrada sobre todas las iteraciones Grasp se devuelve como la solución aproximada. El algoritmo general de Grasp, como mostrado a seguir, considera un conjunto de elementos E condición de parada C, la función objetivo f a optimizar, y un parámetro de relajación $\alpha \in [0,1]$,encontrando la mejor solución s. Mayores detalles consultar en [1].

Grasp (Elementos E, Condicion C, Objetivo f, Relajación)

1. Sea s := solución inicial
2. Mientras (C), hacer
2.1 %:= Construcción(E; f;∝) // una posible solución
2.2 %:= Mejoria(%; f) // mejora solución
2.3 Si % mejor que s, entonces
2.3.1. Registrar solución s := %
2.3. Fin-si
3. Fin-mientras
4. solución Grasp s.

Dada una grilla de n celdas y m objetos, considerados, en peor de los casos, que cada objeto ocupe únicamente una celda de la grilla. Así, en una iteración el Grasp genera similar número de novo que una simple branch de Sears. O sea, en el nivel inicial serán generados n posibles nodos; a partir de una de esos nodos, se generan n -1 posibles nodos en un segundo nivel; en ese proceso, en un nivel máximo m se generan n-m+1 de posibles configuraciones. El total de los posibles nodos generados es dado por:

$$\sum_{k=1}^{m} (n-k+1) = m(2n-m+1)/2$$

Ese número será multiplicado por el número de iteraciones realizadas por Grasp hasta convergir al valor óptimo. Por ejemplo, si $n=25$ celdas y $m=16$ objetos, entonces en peor de los casos, cuando cada uno de los m objetos entran únicamente en una celda, se generan 280 posibles nodos. Si se considera, como en el enfoque formulado en este trabajo, un objeto puede ocupar más de una celda, entonces el número de posibles nodos r generados en una pasada Grasp es $r<280$. Por tanto, en p iteraciones, serán generados $r \times p < 280p$ posibles nodos.

4.2. Grasp en el diseño de la GUI

La estructura Grasp general para el problema en estudio mantiene las mismas componentes de

construcción, mejoría y registro de la mejor solución, tal como mostramos a seguir en el algoritmo GraspGui. Se han considerado como parámetros: el conjunto de objetos *O* (con características dadas en la Tabla 1), la grilla *G* con sus dimensiones, la lista de tareas *J* (con características dadas en la Tabla 1), una condición de parada *C*, y el parámetro de relajación $\alpha \in [0,1]$ La condición de parada *C* usada, es dada por una condición que verifica si antes de *C* iteraciones del GraspGui la mejor solución encontrada ha cambiado, esto es el algoritmo termina cuando hay una convergencia de la solución en al menos *C* iteraciones.

GraspGui (Objetos O, Condicion C,
Objetivo f, Grilla G, Tareas J, Relajacionα)

1. Sea s := diseño propuesto // configuración inicial
2. Mientras (*C*), hacer
2.1 $\tilde{s}$:= GraspConstructionGui (O; G; f; J; α) // solución
2.2. $\tilde{s}$:= GraspMejoriaGui ($\tilde{s}$; G; f) // mejora solución
2.3. Si $\tilde{s}$ mejor que s, entonces
2.3.1. Registrar solución s := $\tilde{s}$
2.3. Fin-si
3. Fin-mientras
4. solución optima *s*.

Al final de cada iteración del GraspGui (pasos 2. - 3.) se registra en *s* la mejor solución encontrada entre la solución obtenida por la GraspConstructionGui y su solución mejorada dada por GraspMejoriaGui. La solución *s* obtenida al final del algoritmo (paso 4.), representa una configuración de la GUI y será considerado como un diseño óptimo para su evaluación.

4.2.1. Grasp construcción en GUI

El GraspConstructionGui construye una solución en forma iterativa; en cada iteración del algoritmo (pasos 2. - 3.) se ubica una celda para cada objeto, usando un criterio goloso-aleatorio respecto al costo de la tarea y la frecuencia relativa.

Considere que se desea ubicar un objeto *Ok*. Se entiende que en la iteración anterior *(K- 1)* existe una configuración establecida, definida por el diálogo, los *K-1* primeros objetos y algunas celdas disponibles. El criterio Grasp (goloso-aleatorio) usado consiste en lo siguiente: primero se determina los costos asociados de ubicar el objeto *Ok* en cada una de las celdas disponibles (paso 2.1.); segundo se determina el menor y mayor costo, denotado por β_{min} y β_{max} respectivamente (pasos 2.2. y 2.3.); tercero se construye la lista de candidatas restrictas *(RCL)* siendo una lista de posibles ubicaciones para el objeto *Ok* y que presentan un costo asociado entre β_{min} y una relajación de este $\beta_{min} + \alpha(\beta_{max} - \beta_{min})$; finalmente, la ubicación para el objeto *Ok* es dado por una ubicación obtenida desde RCL a través de una selección aleatoria.

El proceso anterior es realizado para todos los objetos $\{O_k\}_{k=1,\ldots,n}$ de la lista de objetos y, por tanto, será realizado *n* iteraciones. El número de posibles configuraciones disminuye cada vez que el número de iteraciones se incrementa; además este número también depende del tamaño del objeto en relación del tamaño de las celdas. Por otro lado, podemos notar que en cada iteración de la GraspConstructionGui se construye, en forma aleatoria-golosa, una ramificación del árbol de configuraciones de Sears.

A continuación, mostramos el procedimiento construcción de Grasp Grasp ConstruccionGui, siendo *S* la configuración generada por este.

GraspConstructionGui (Objetos O, Grilla G, Tarea J, Relajacion α)

1. $S = \phi$; // mejor solución inicializada en vacío
2. Para $\{O_k\}_{k=1,\ldots,n}$ hacer // para cada objeto hacer
2.1. Generar costos $\{C_p\}$ de posibles configuraciones para O_k , usando J y celdas disponibles de G
2.2. $\beta_{min} = \min\{C_p\}$ // define costo menor
2.3. $\beta_{max} = \max\{C_p\}$ // define costo mayor
2.4. $RCL = \{p \in G : \beta_{min} \le c_p \le \beta_{min} + \alpha(\beta_{max} - \beta_{min})\}$ // Conjunto de mejores soluciones
2.5. $p_k = Random\{RCL\}$ // Posición aleatoria de RCL
2.6. Colocar O_k en posición p_k de la grilla G
2.7. $S = S \cup \{O_k, p_k\}$ // construye solución
3. Fin-para
4. Retornar(*S*).

Observe que en la iteración *k*, el costo computacional es *O(n-k)* operaciones, pues para el *k*-ésimo objeto a ubicar sólo hay *n-k* posiciones. Siendo *m* el total de objetos a ubicar, la complejidad asintótica del algoritmo GraspConstrutionGui es dado por O(m(n-m)

4.2.2. Mejoría en GUI

En cada ejecución de GraspConstrutionGui se genera una configuración que en su proceso iterativo de construcción sólo considera los objetos ya ubicados. Esto es, consideremos por un instante que en la interacción *r* del proceso de construcción, el objeto *Or* se posiciona en una celda apropiada *Pk* que sólo contempla los *r*-1 objetos ya ubicados en las *r*-1 iteraciones anteriores, no contemplando los *m-r* objetos restantes.

En este sentido, proponemos un procedimiento de mejoría que en cada iteración reubique un objeto que presente menor costo respecto a la solución que se desea mejorar. Este procedimiento se puede repetir mientras exista una mejor solución.

En cada iteración del GraspMejoriaGui (segmentos de pasos 2.-3.), el algoritmo determina una mejor solución respecto a la configuración de la iteración anterior. En el segmento 2.2-2.3 se busca el objeto de menor costo en función de los *m*-1 objetos restantes.

GraspMejoriaGui (Solución S, Grilla G, Tarea J)

1. C = Costo(S) // costo de la solución
2. Hacer
2.1. $C_a = C$ // actualiza costo anterior
2.2. Para $O_i \in S$, i=1,..., m // para cada objeto que define S hacer
2.2.1. Retirar temporalmente O_i de S
2.2.2. Buscar la ubicación de menor costo para O_i
2.2.3. Si $Costo(O_i) < C$, entonces
2.2.3.1. $C := Costo(O_i)$
2.2.3.2. $k := i$, $p :=$ ubicación de menor costo para O_i
2.2.4. Fin-Si
2.3. Fin-Para
2.3. Si $C < C_a$, entonces
2.3.1. $S := S - \{(O_k, p_k)\} - \{(O_k, p)\}$ // actualizar S con nueva ubicación para O_k
3. Hasta $(C_a > C)$
4. Retornar(Solución Mejorada S)

En la Figura 4 se presenta la configuración obtenida de aplicar nuestra propuesta de mejoría sobre la solución descrita en la Figura 2. Observe que el costo de la configuración se ha reducido de 51.100 para 50.03.

5. Testing de eficiencia

Para calibrar el parámetro de relajación α para el ejemplo de la Figura 2, se ha realizado varias corridas del algoritmo GraspConstrutionGui con 100, 500, 1000 y 2000 iteraciones con distintos valores de α(0.1,0,15, 0.2 y 0.3). Los resultados numéricos mostrados en la Tabla 3, indican que el mejor valor para alfa es 0.2, obteniendo la configuración ilustrada en la Figura 5.

Consideramos dos aspectos en la eficiencia del método propuesto: Número de nodos generados por el proceso; y la medida del diseño apropiado. Para eso, usamos el ejemplo de la Tabla 1 que describe las tareas envolviendo 5 objetos de la Tabla 2, cuyo diseño propuesto es ilustrado por la Figura 2. En este caso, es calculada una grilla de 6x7=42 celdas.

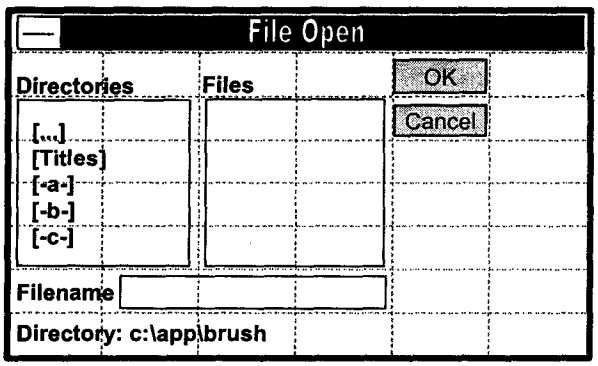

Figura 4: Una GUI generada con Grasp mejoría.

α	Corridas			
	100	500	1000	2000
0;1	51.70	51.60	51.30	51.23
0;15	51.68	51.60	51.40	51.10
0;2	51.01	50.98	50.94	50.03
0;3	51.10	51.04	50.01	50.98

Tabla 3: Resultados numéricos para la calibración de a para GUI ejemplo.

Los cinco objetos del ejemplo ocupan un total de 30 celdas, ya que tres de ellos ocupan celdas múltiplas.

Si cada celda ocupada fuese considerada como posición de un objeto, entonces, usando (5), se tendrían 835 posibles nodos generados, pero por el método propuesto ese número se decrementó considerablemente, tal como muestra la Tabla 4.

Grilla	Objetos		Nodos			Iteraciones
	Básico	Reales	Posibles	Generados	Total	
6x7	10	5	231450	11572	243022	15000
6x7	10	5	251258	12562	263820	20000
8x8	9	6	316368	15818	332186	15000
8x8	9	6	382369	19118	401487	20000

Tabla 4: Grilla, objetos y nodos generados en la construcción de una GUI por Grasp.

Aplicamos la expresión (1) para el ejemplo usado en la ilustración de las etapas del diseño apropiado con Grasp. El costo de la GUI propuesta, calculada usando la Tabla 1 de tareas del usuario, es CGP=68,20 y el costo de la GUI mejorada, que consideramos próxima a la óptima, es CGO=51,21. Así, el índice del diseño propuesto es

$$DA = \frac{51.01}{68.20} \approx 0{,}75$$

Que indica que el diseño propuesto no es óptimo. También calculamos el costo de diseño usando un algoritmo goloso miope, basado en el método de Sears, dando como resultado CGO=51,10, que comparado con 51,01 de GUI mejorada con el Grasp es mayor. Siendo así, el diseño apropiado del método goloso de Sears respecto al método Grasp es $DA = 51.01/51.100 \approx 998$, que indica que el diseño de la GUI por método goloso se aproxima un poco al diseño de la GUI por método Grasp.

En la Tabla 5 mostramos el número de corridas C para α=0,2 el costo de GUI generada por la Grasp Construcción aplicada al ejemplo usado con cinco objetos. Observamos, el tiempo de ejecución del método en la fase de Grasp construcción disminuye un poco, por no decir que es constante, cuando incrementa número de iteraciones del proceso Grasp.

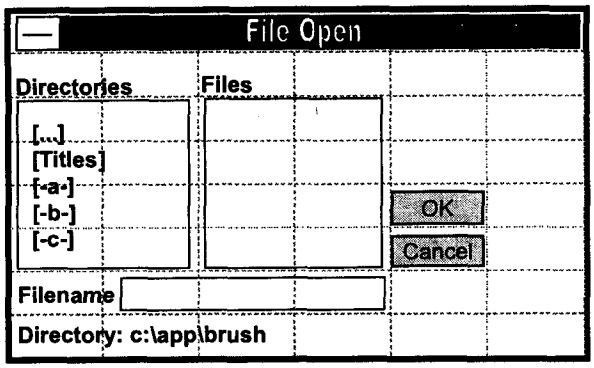

Figura 5. Una GUI generada por la Grasp Construction.

Método	α	N. Corridas	Costo (seg.)
Garsp-Construccion	0.2	$100 \leq C \leq 1000$	51.01
	0.2	$1000 \leq C \leq 10000$	50.04
	0.2	$1000 \leq C \leq 20000$	50.01
Sears	-	-	51.10

Tabla 5. Ejecuciones del Grasp-Construccion y Sears para GUI ejemplo.

6. Conclusiones y trabajos futuros

Se ha propuesto un método de redistribución eficiente de objetos de una GUI, inspirado en el método Grasp. Los resultados numéricos como resultado visual y tiempo de ejecución son buenos respecto al método de Sears. En nuestro testing y validación del método hemos usados un ejemplo de una GUI compuesto de cinco objetos y una descripción de ocho tareas del usuario. Desde luego que, los dados usados para esta validación son objetos sintéticos, descritos por sus parámetros geométricos y valores numéricos. Consideramos que, un testing de una GUI real no está distante del ejemplo usado, pues teniendo un módulo que identifique y capture los objetos de una interfaz real de cualquier forma se tendría que asociar con valores numéricos y parámetros geométricos respectivos. Esta fase, será nuestro objetivo inmediato, como trabajo futuro, así mismo, asociar a la lista de tareas los principios de semiótica, estudiados ampliamente en [14,15,16], para una buena distribución de los objetos.

Referencias bibliográficas

[1] Feo T., Resende M. Greedy Randomized Adaptive Search. Procedure Journal of Global Optimization, n. 6, 1995, pp. 109-133.

[2] Sears, A. Layout Appropriateness: A Metric for Evaluating User Interface Widget Layout. IEEE Trans. Software Engineering, vol. 19, n.7, 1993, pp 707-719.

[3] Casner, S. y Larkin, J. H. Cognitive Efficiency Considerations for Good Graphic Design. Cognitive Science Society Proceedings, 1989.

[4] Kieras, D. Towards a Practical GOMS Model Methodology for User Interface Design. M Helander (Ed.), Handbook of Human-Computer Interaction. Amsterdam. Elsevier Science Publishers, 1988.

[5] Tullis, T. A system for evaluating screen formats: Research and application. In: R. Hartson & D. Hix (Eds.), Advances in Human-Computer Interaction, vol. 2, 1988, pp. 214-286.

[6] Perlman, G. An Axiomatic Model of Information Presentation. Proceedings of the 31st Annual Meeting of the Human Factors Society, 1987, pp. 1129-1233.

[7] Lohse, J. A Cognitive Model for the Perception and Understanding of Graphics. CHI'91 Proceedings, 1991, pp. 137-144.

[8] Sears, A. AIDE: A tool to assist in the design an evaluation of user interfaces, Depaul University, Chicago, 1999.

[9] Linz, T. y Daigl, M. ESSI PIE 24306, imbus GMBH, D-91096 Möhorendorf, Germany.

[10] Fenton, N. Software Metrics, Chapman Hall, 1991.

[11] Szekely, P. Template-Based Mapping of Application Data to Interactive Displays. UIST'90 Proceedings, 1990, pp. 1-9.

[12] MacKenzie, I. S. y Buxton, W. Extending Fitts' Law to Two-Dimensional Tasks. CHI'92 Proceedings, 1992, pp. 219-226.

[13] Feo,T.A. y Resende, M.G.C. A probabilistic heuristic for a computationally difficult set covering problem. Operations Research Letters, 1989, 8:67-71.

[14] Andersen, P.B. A Theory of Computer Semiotics. Cambridge, MA. Cambridge University Press, 1990.

[15] Nadin, M. Interface design and evaluation. Advances in Human-Computer Interaction, vol. 2. Norwood, NJ: Ablex Publishing Corp., 1988.

[16] de Souza, C.S. Knowledge Based Systems. Amsterdam, v.14, n.8, 2001, p.415-418.